TD2 : Gestion des transactions et de la Concurrence

Exercice 1 :

L'exécution des transactions dans les cas A,B,C et D sont susceptibles de générer des problèmes. Préciser la nature du problème.

A.

T_1	T_2	BD
		$A = 10$
read A		
	read A	
A = A + 10		
write A		$A = 20$
	A = A + 50	
	write A	$A = 60$

B.

T_1	T_2	BD
		$A + B = 200$
		$A = 120$ $B = 80$
read A		
A = A - 50		
write A		$A = 70$
	read A	
	read B	
	display A + B (150 est affiché)	
read B		
B = B + 50		
write B		$B = 130$

C.

T_1	T_2	BD
		$A = 50$
	A = 70	
	write A	$A = 70$
read A (70 est lu)		
	rollback (La valeur initiale de A est restaurée)	$A = 50$

D.

T_1	T_2	BD
		$A = 10$
	read A (10 est lu)	
A = 20		
write A		$A = 20$
	read A (20 est lu)	

Exercice 2 :

Supposons une table *T(id, valeur)*, et la procédure PL/SQL suivante qui copie la valeur d'une ligne vers la valeur d'une autre :

```
/* Une procédure de copie */
create or replace procedure Copie (id1 INT, id2 INT) AS
 -- Déclaration des variables
 val INT;
 BEGIN
   -- On recherche la valeur de id1
   SELECT valeur INTO val FROM T WHERE id = id1

   -- On copie dans la ligne id2
   UPDATE T SET valeur = val WHERE id = id2

   -- Validation
   commit;
 END;
/
```

On prend deux transactions c*opie(A, B)* et *copie(B,A)*, l'une copiant du nuplet *A* vers le nuplet *B* et l'autre effectuant la copie inverse. Initialement, la valeur de *A* est *a* et la valeur de *B* est *b*.

1. Qu'est-ce qui caractérise une exécution concurrente correcte de ces deux transactions ?
   A. A et B valent a
   B. A et B valent b
   C. A et B ont la même valeur
   D. A vaut b et B vaut a
2. Voici une exécution concurrente de *Copie(A, B)* et *Copie(B,A)* : $l_1(A)l_2(B)e_1(B)e_2(A)$

   En supposant qu'elle s'exécute sans contrôle de concurrence, avec une valeur initiale de *A* à *a* et une valeur de *B* à *b*, quel est l'état de la base à la fin ?

   A. A=a et B=b
   B. A=a et B=b
   C. A=b et B=a
   D. A=b et B=b
3. Cette exécution est-elle sérialisable?

Exercice 3 :

Supposons qu'un hôpital gère la liste de ses médecins dans une table (simplifiée) *Docteur(nom, garde)*, chaque médecin pouvant ou non être de garde. On doit s'assurer qu'il y a toujours au moins deux médecins de garde. La procédure suivante doit permettre de placer un médecin au repos en vérifiant cette contrainte.

```
/* Une procédure de gestion des gardes */

create or replace procedure HorsGarde (nomDocteur VARCHAR) AS

 -- Déclaration des variables
 val nb_gardes;

 BEGIN
   -- On calcule le nombre de médecin de garde
   SELECT count(*) INTO nb_gardes FROM Docteur WHERE garde = true

   IF (nb_gardes > 2) THEN
      UPDATE Docteur SET garde = false WHERE nom = nomDocteur;
      COMMIT;
   ENDIF
 END;
/
```

Supposons que nous ayons trois médecins, Philippe, Alice, et Michel, désignés par *p*, *a* et *m*, tous les trois de garde. Voici une exécution concurrente de deux transactions T1= *HorsGarde("Philippe")* et T2=*HorsGarde("Michel")*.

$$l_1(p)l_1(a)l_1(m)l_2(p)l_2(a)l_2(m)e_1(p)e_2(m)$$

1. Quel est, avec cette exécution, le nombre de médecins de garde constatés par T1 et T2 ?
   a. 3 pour T1, 2 pour T2
   b. 3 pour T1, 3 pour T2
   c. 2 pour T1, 3 pour T2

2. Quel est le nombre de médecins de garde après la validation de T1 et T2 ?

    a. 2
    b. 1
    c. 0

Exercice 4 :

On reprend les transactions de gestion de docteurs et de leurs gardes de l'exercice 2. On a l'exécution concurrente suivante des deux transactions cherchant à lever la garde de Philippe et de Michel.

$$l_1(p)l_1(a)l_1(m)l_2(p)l_2(a)l_2(m)e_1(p)e_2(m)$$

1. Trouver les conflits
2. En déduire (argumenter) que cette exécution concurrente n'est pas sérialisable
3. Appliquer l'algorithme 2PL (Verrouillage à deux phases).

Exercice 5 :

Identifiez les conflits et construisez les graphes de sérialisabilité pour les exécutions concurrentes suivantes. Indiquez les exécutions sérialisables et vérifiez s'il y a des exécutions équivalentes.

H1 : $e_2[x]\ e_3[z]\ e_2[y]\ c_2\ l_1[x]\ e_1[x]\ c_1\ l_3[y]\ c_3$
H2 : $l_1[x]\ e_2[y]\ l_3[y]\ e_3[z]\ c_3\ e_1[z]\ c_1\ e_2[x]\ c_2$
H3 : $e_3[z]\ e_1[z]\ e_2[y]\ e_2[x]\ c_2\ l_3[y]\ c_3\ l_1[x]\ c_1$

Exercice 6 :

1. Vérifier la sérialisabilité des exécutions ci-dessous.
2. Pour chacune des exécutions ci-dessous, décrire les transactions qui seraient placées en attente dans le cas d'un verrouillage deux phases. (On ne considèrera pas la suite des actions d'une transaction mise en attente)
3. Avec un système d'estampillage, quelles transactions seraient défaites ?

A) r1(x), w1(x), r2(z), r1(y), w1(y), r2(x), w2(x), w2(z)
B) r1(x), w1(x), w3(x), r2(y), r3(y), w3(y), w1(y), r2(x)
C) r1(x), r2(x), w2(x), r3(x), r4(z), w1(x), w3(y), w3(x), w1(y), w5(x), w1(z), w5(y), r5(z)

Exercice 7 :

Soit *x* et *y* deux articles de la table *R* et l'exécution suivante :
H : $l_1[R]\ l_2[R]\ e_1[x]\ l_3[x]\ e_3[x]\ c_2\ e_1[y]\ l_3[y]\ c_1\ e_3[y]\ c_3$

Indiquez l'ordre d'exécution établi par un ordonnanceur avec verrouillage hiérarchique. Considérez que l'on utilise un verrou SIX si une transaction lit une table pour modifier ensuite des articles de celle-ci.